

Quality Assurance for Taxonomic Database Working Group Standards and Software

Introduction

There are hundreds of biological databases around the world in museums, herbariums, universities, government, and non-government organizations. With the problems of the decline in biodiversity and invasive species there is an increasing need to make this data available to a wide audience for research and decision making. The Taxonomic Database Working Group (TDWG) produces standards and software toolkits for the storage and exchange of biological data. Introducing quality assurance practices for the TDWG can greatly increase the utilization of its products while reducing overall development costs and keeping support costs at a minimum. This transition can be equated to the development of web search engines, such as Google, which greatly increased the access to information on the Internet.

Standards and software toolkits, both referred to herein as “products”, can fail to achieve their goals due to (1) excessive complexity, (2) features changing until resources are exhausted (feature creep), (3) products that do not match user’s needs, (4) user’s being unable to obtain help when they encounter problems, or (5) updates are not made to repair defects and maintain system compatibility. Causes 1 and 2 are examples of issues with managing the definition of the product, 3 is a problem in implementation and testing, while 4 and 5 are maintenance issues.

A formal software lifecycle defines the process of creation of the software and critical steps to minimize the problems mentioned above. It has been repeatedly shown that moving to a formal software process not only improves the quality of products but also reduces overall development costs.

Definitions

Users of standards and software products from the Taxonomic Database Working Group (TDWG) include individuals who use a standard protocol, software toolkit, or the resulting system. This includes scientists, researchers, software developers, and IT professionals.

Stakeholders include funding organizations and TDWG executives.

The product includes protocol standards, software toolkits, and associated documentation.

Software Lifecycle Phases

Software lifecycles are typically divided into phases of development such as the following.

- **Investigation** – Determining the products requirements and feasibility
- **Design** – Defining the components of the product and their characteristics

- **Implementation** – Creating the components and integrating them into the final product
- **Testing** – Insuring the product meets the requirements
- **Delivery** – Providing marketing, training, and the product to users
- **Maintenance** – Insuring customer issues are resolved

The focus of the investigation phase is to determine the requirements of the product. This includes surveys, interviews, and analysis of data from the target users to insure the product meets their requirements. Use Cases are one method of capturing user requirements and document specific things the users expect to be able to with the product. Investigation also includes any technical feasibility studies required to insure the product can be constructed as required. The investigation phase is also used to determine the resources available and the schedule required for completion.

The design phase includes breaking the product into its constituent components, defining the characteristics of each component, and the interfaces between the components. This is continued until each component is specified to the level that the amount of time can be determined for its completion.

The implementation phase is when each component of the product is created and then integrated with other components to create the final product. During investigation there are usually discoveries that cause the design to be updated.

The purpose of the testing phase is to insure that each component and the overall product meet the requirements defined during investigation. This typically includes; unit testing of each component of the system, automated testing, and testing of use cases. Tracking defects during testing insures that each defect found is resolved appropriately.

Delivery consists of creating training and marketing materials and making the product available to the users.

Maintenance consists of support to end-users and periodic updates to the product. Each update to the product should be managed as a small development project with its own deliverables.

Phases of a project can last for years or a day based on the complexity of the project. Likewise the deliverables should be adjusted.

Schedule Methodology

The two dominant methods for managing a development schedule are waterfall and spiral. The waterfall method expects each phase of the schedule to be completed before the next is started. Thus the design would be completed before any implementation is begun. This provides clear measures of progress in the development of the software but when problems arise that were not foreseen the project must return to the previous phase and effectively start over. Spiral models execute all phases simultaneously providing a very high level of feedback between the phases but making it difficult to measure progress. The Stair Step method developed by Rick Lesser and Jim Graham at Hewlett-

Packard is a more general approach that allows the amount of overlap between each phase to be adjusted based on the needs of the project. Thus there would be significant overlap between each phase allowing for feedback between say design and implementation but each phase is completed sequentially so progress can be measured.

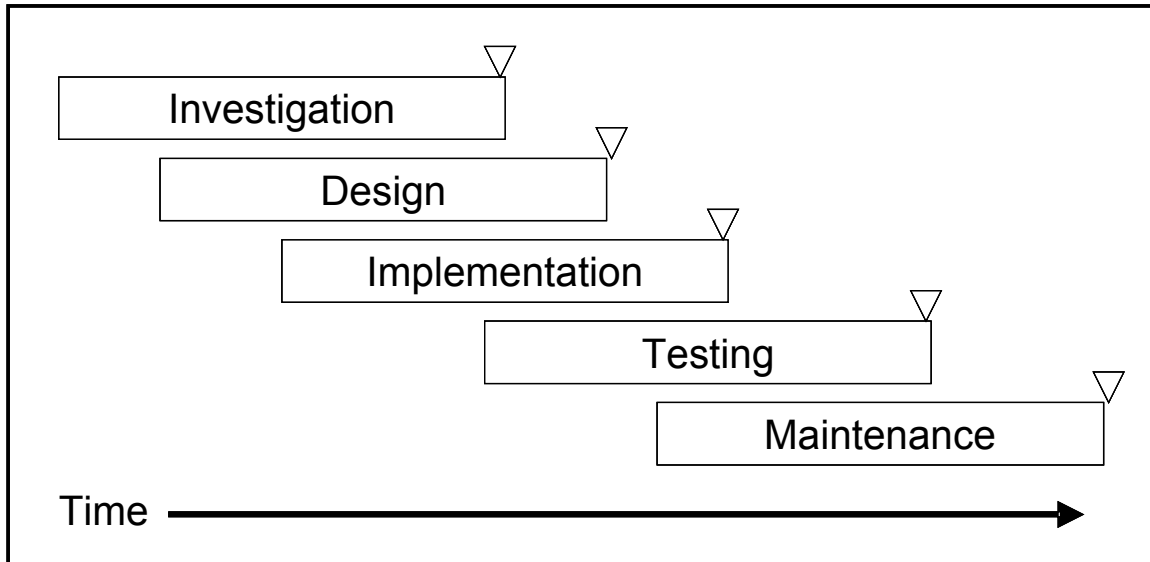


Figure 1. Stair-Step software lifecycle

Flow of Information

The flow of information within a development product begins by interviewing users to develop Use Cases that describe how they will use the product (Figure 1). Analysis of the Use Cases will then resolve some cases with existing products while other cases can be resolved with a combination of existing products and the product under consideration. The resulting features are added to the Requirements Specification along with other information such as system requirements, performance requirements, compatibility requirements, and the user's level of expertise.

The Design of the product is based on the Requirements Specification and guides implementation decisions to insure the product features meets the user's needs. The Test Plan is also based on the Requirements Specification and guides the Testing to insure the features, performance, compatibility, and robustness of the product meets the user's needs.

Defects from internal Testing and from Users are used to improve the product through fixing defects and by improving the Requirements for future releases.

This circular flow provides a continuous product improvement cycle and keeps everyone involved focused on meeting the user's needs.

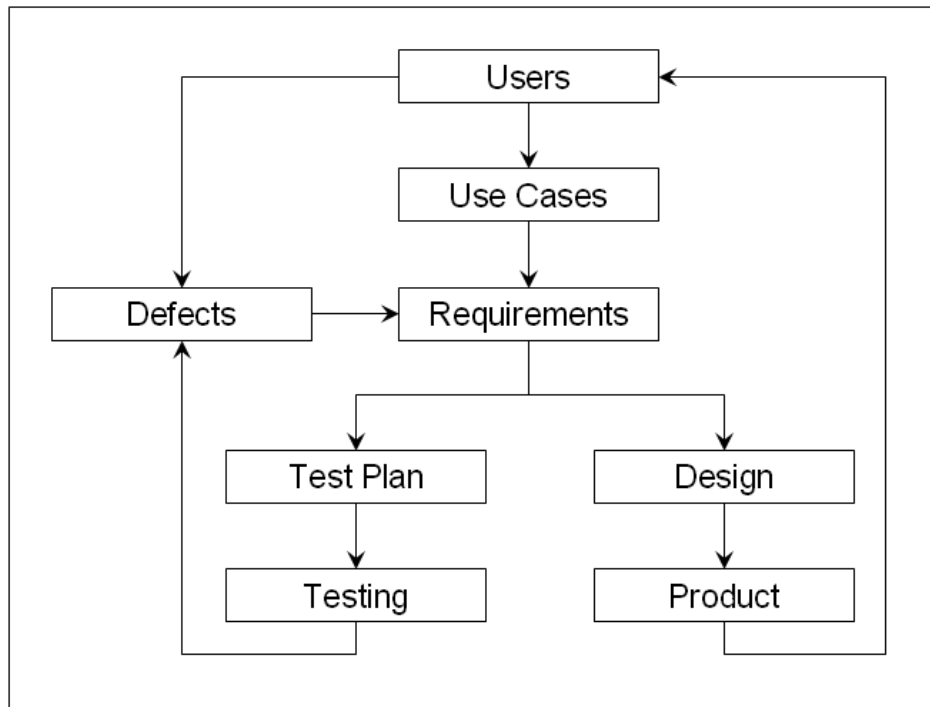


Figure 1. Simplified software lifecycle flow diagram

Software Lifecycle Deliverables

Below are the minimal deliverables for a software lifecycle. For small product many of the deliverables can be very short documents. The most critical items to insure product quality are the Requirements Specification, Testing, and Customer Support that provides feedback into the design and implementation phases of future releases.

1. Investigation

- **Requirements specification:** Insures the product meets the needs of stakeholders and users.
- **Use cases:** Test the product against key use scenarios as development progresses
- **Feasibility studies:** Assure the required robustness and performance can be met before development progresses too far to change the direction if needed
- **Alternatives analysis:** Insures the best technologies are selected for implementation

2. Design

- **Product design:** The goal of the design is to create the simplest solution that meets the requirements thus insuring the best chance of success with minimum resources
- **Test plan:** Insures the components and completed product meets the requirements
- **Maintenance plan:** Insures the product can be updated within resource constraints

3. Implementation

- **Product:** The components assembled into a final product
- **Documentation:** Tutorial and reference material on how to use the product
- **Test tools:** For in-house testing of the product

4. Testing

- **Defect Database:** Insures that defects are tracked until resolved appropriately. Also provide information for users and support staff on the problems in each release.
- **Test metrics:** Metrics including the number of defects found and resolved over time, provide a metric on the quality of the product helping to make decisions on releases and additional testing.

5. Delivery

- **Training materials:** Materials for users, support staff, and anyone marketing the product.

6. Maintenance

- **Defect Database:** Continues to be updated with additional defects from users and resolutions.
- **Test metrics:** Provide a metric on each new release.
- **Upgrades:** Insure the product continues to meet user needs including operating system compatibility and new features.

References
